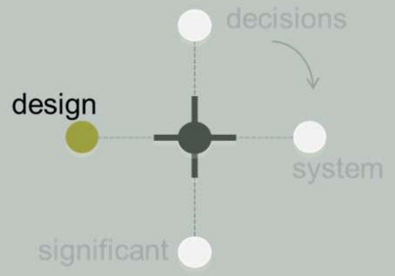
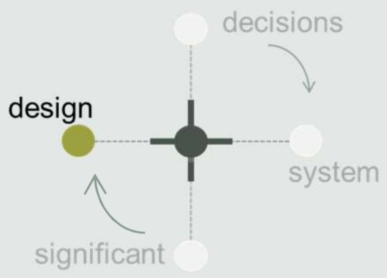
System Design



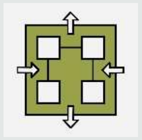
Design in Context

Design as Theory Building

Frames and Practices (or what's ahead)



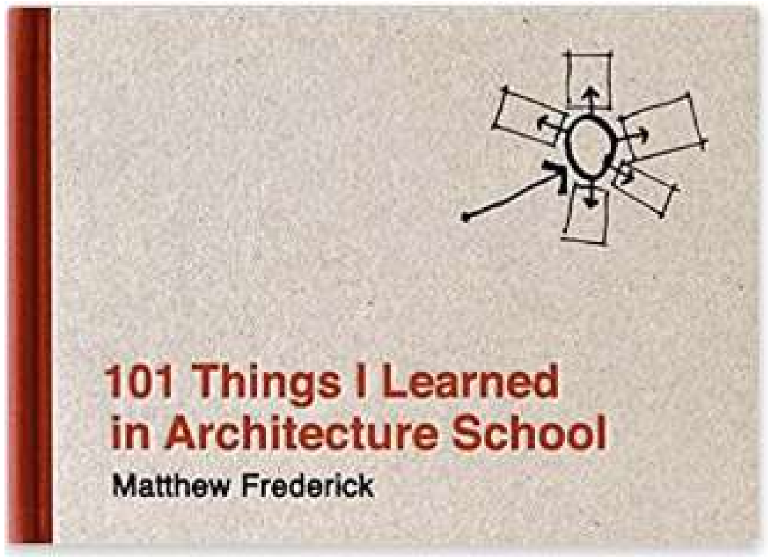
- Design!
- In next larger context

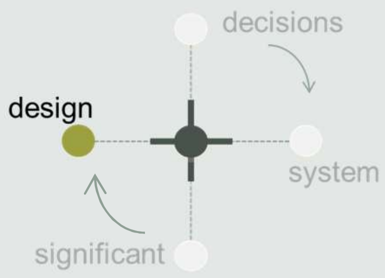


Design in Context

"Always design a thing by considering it in its next larger context."

— Eliel Saarinen





- Design!
- In next larger context
- Context matters

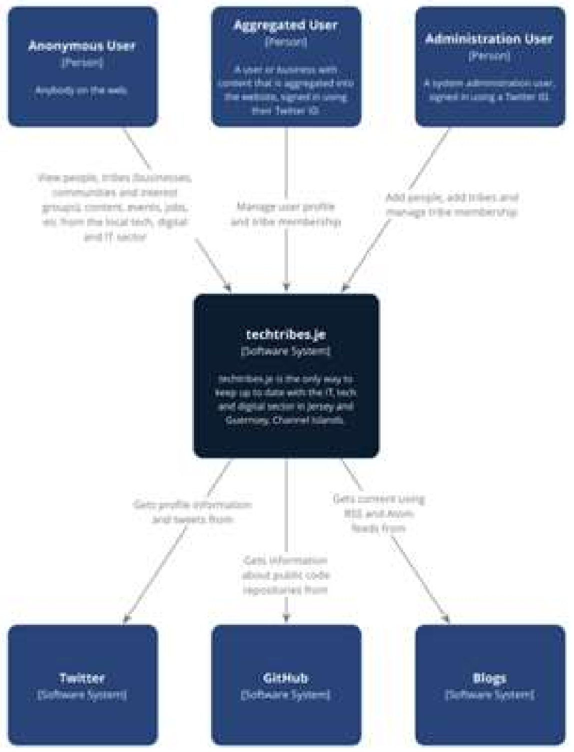
Context Matters

"Design quality is not a property of the code. It's a joint property of the code and the context in which it exists."

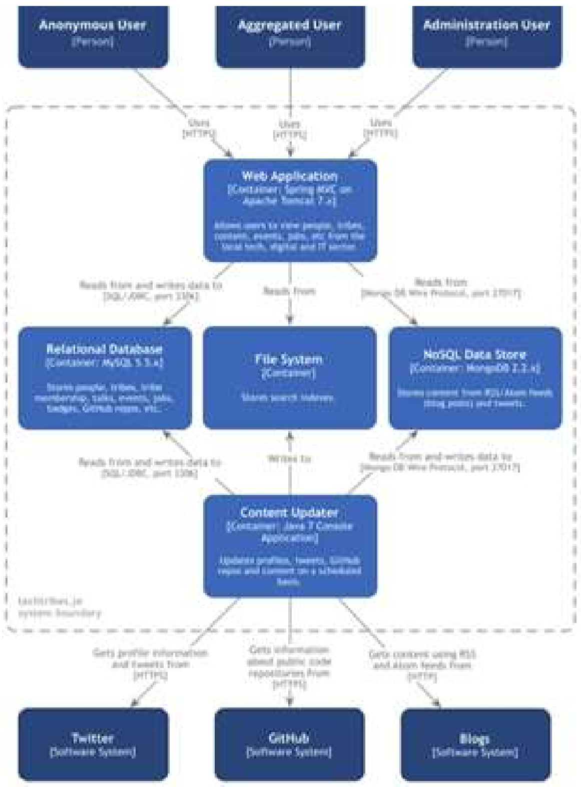
– Sarah Mei

Image source: @sarahmei

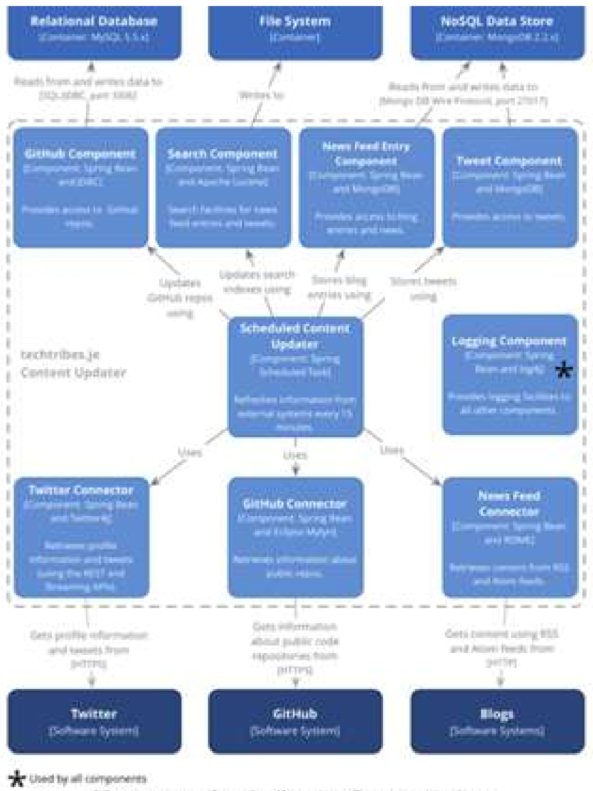
C4: Context, Containers, Component



[System Context] techtribes.je



[Containers] techtribes.je

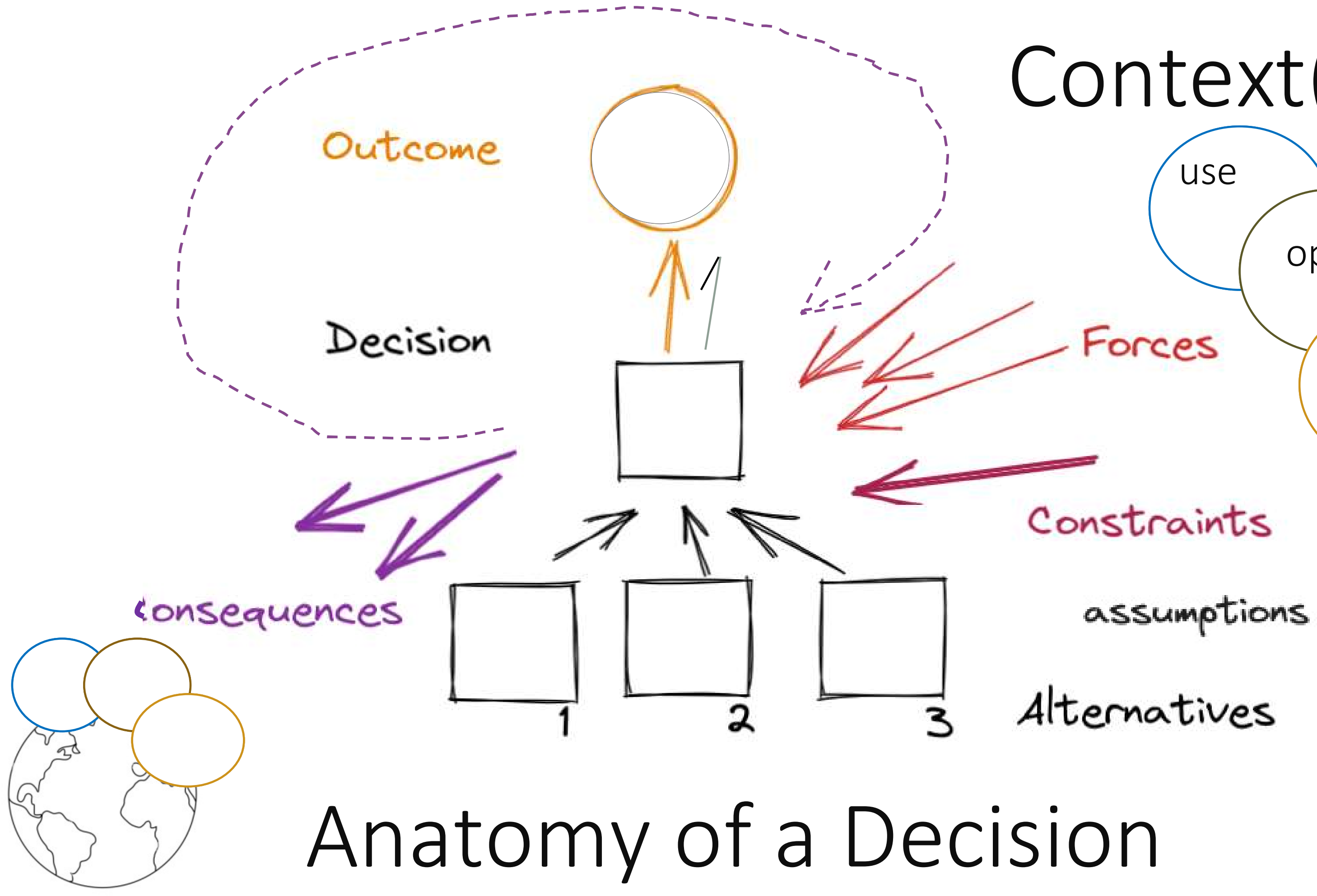


[Components] techtribes.je - Content Updater

Image:
Simon Brown’s C4 Model https://c4model.com/

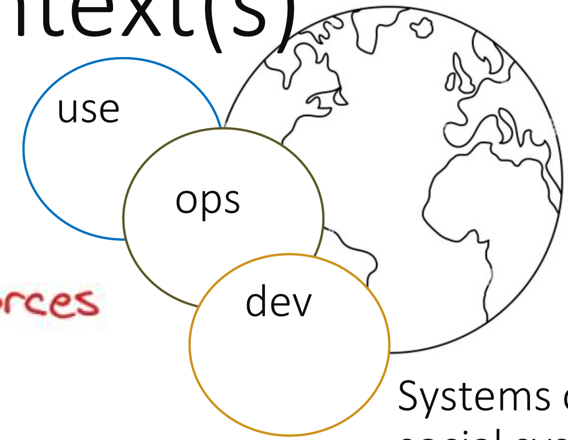
Recall: Forces

Who gains? Who feels the pain? When (e.g., gain now versus pain in a year)?



Anatomy of a Decision

Context(s)



Systems of work; social systems
Competitive landscape

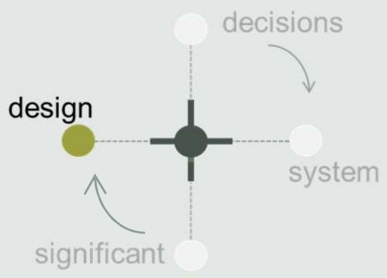
Title: short noun phrase

Context: desired **outcomes** and the **forces** at play (probably in tension) +Assumptions +Aternatives considered

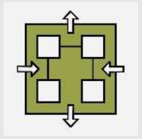
Decision: describes our response to these forces

Status: proposed, accepted, deprecated or superseded

Consequences: describes the resulting context, after applying the decision

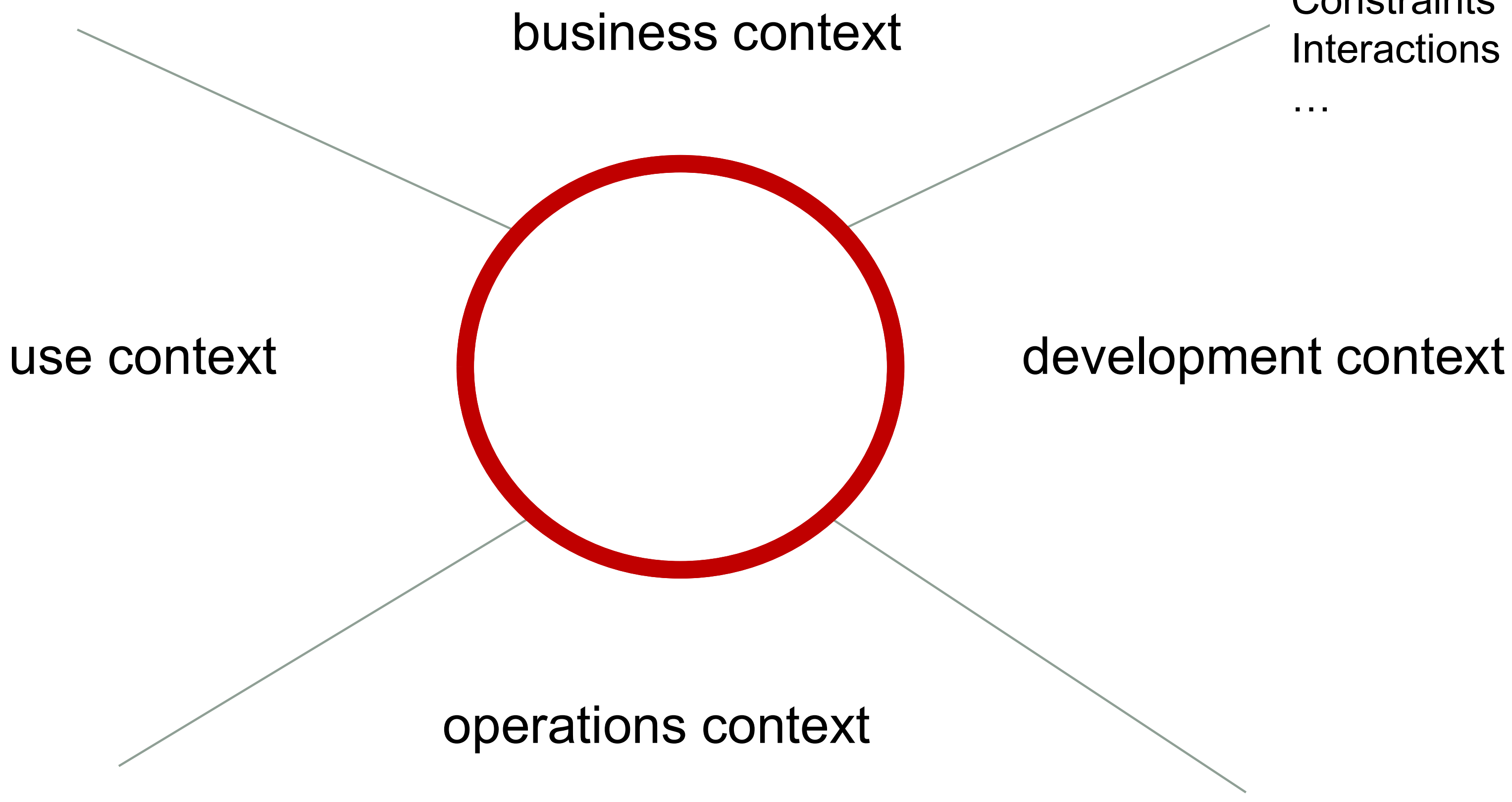


- Design!
- In next larger context
- Context matters
- C4
- Forces



Design in Context

Context:
Needs
Threats
Opportunities
Value flows
Constraints
Interactions
…

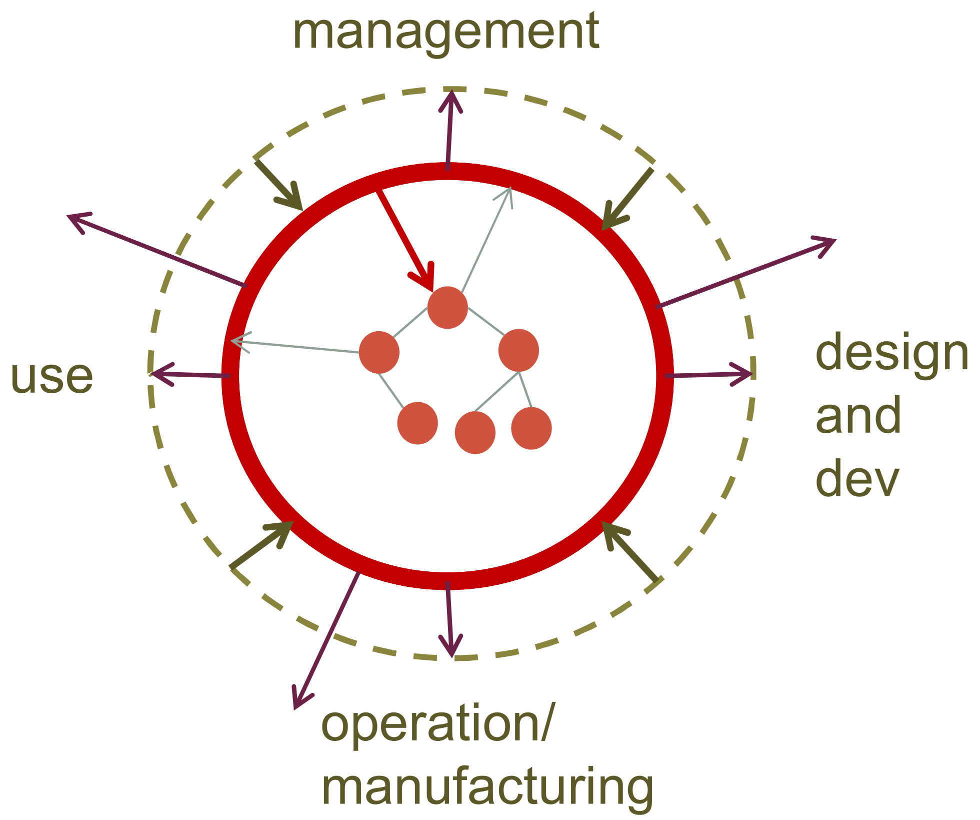


System Design: in Context

Design in context(s)

- Contexts of use, of design and development, of manufacturing and operation, of management
- Social, political, economic, technical contexts

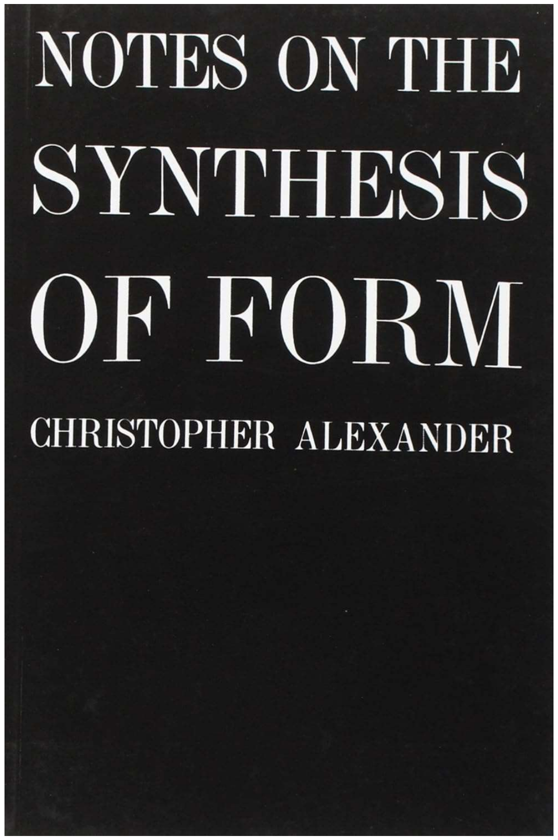
→ move inwards (zoom in), move outwards (zoom out); pan around and scan; surface forces and constraints and consequences



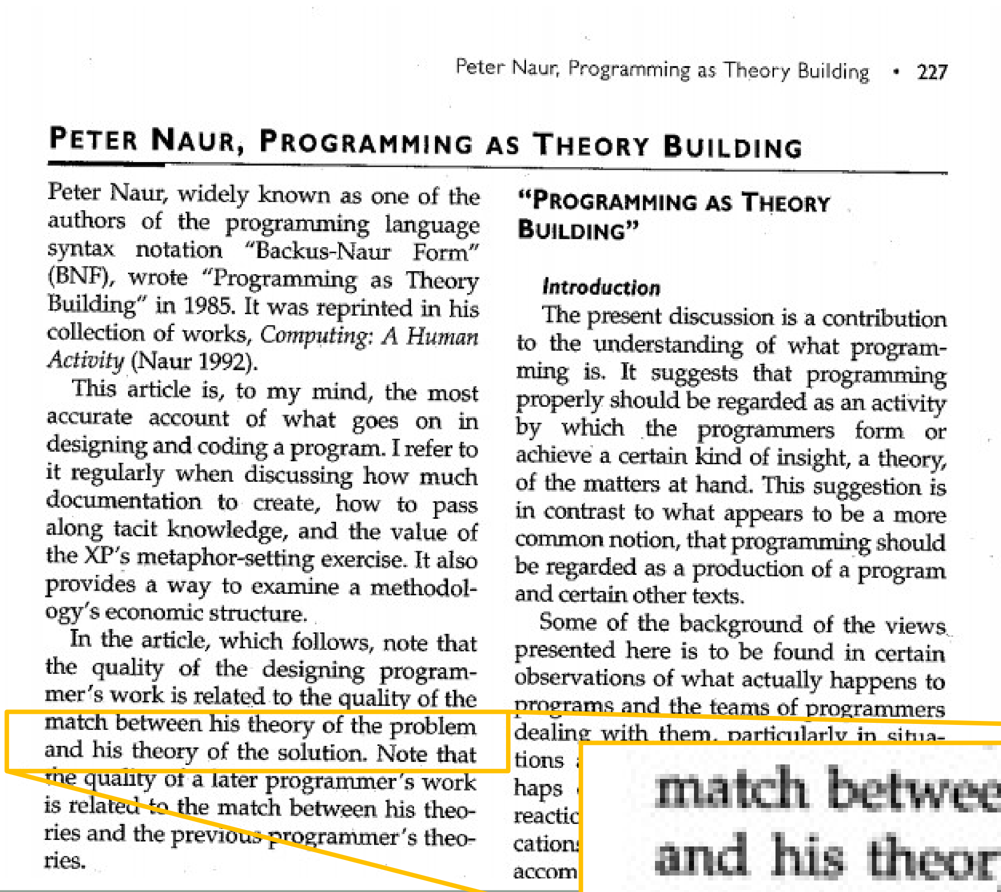
Form and Context

“Every design problem begins with an effort to achieve fitness between two entities: the form and its context. The form is the solution to the problem; the context defines the problem.”

— Christopher Alexander, Notes on the Synthesis of Form, 1964.



Theory Building



Peter Naur, Programming as Theory Building

Peter Naur, widely known as one of the authors of the programming language syntax notation "Backus-Naur Form" (BNF), wrote "Programming as Theory Building" in 1985. It was reprinted in his collection of works, *Computing: A Human Activity* (Naur 1992).

This article is, to my mind, the most accurate account of what goes on in designing and coding a program. I refer to it regularly when discussing how much documentation to create, how to pass along tacit knowledge, and the value of the XP's metaphor-setting exercise. It also provides a way to examine a methodology's economic structure.

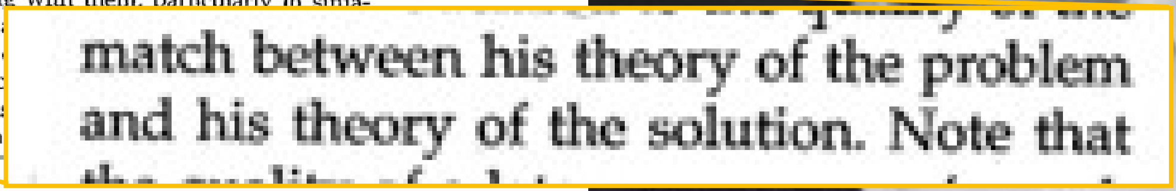
In the article, which follows, note that the quality of the designing programmer's work is related to the quality of the match between his theory of the problem and his theory of the solution. Note that the quality of a later programmer's work is related to the match between his theories and the previous programmer's theories.

"Programming as Theory Building"

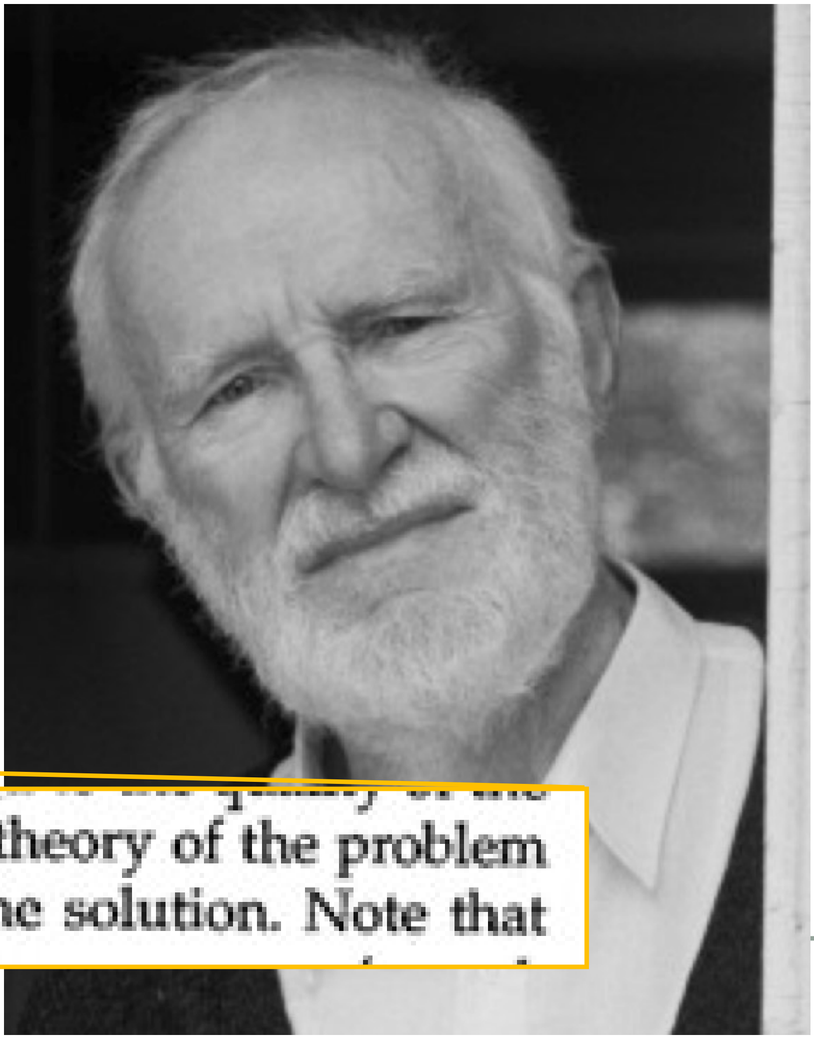
Introduction

The present discussion is a contribution to the understanding of what programming is. It suggests that programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast to what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts.

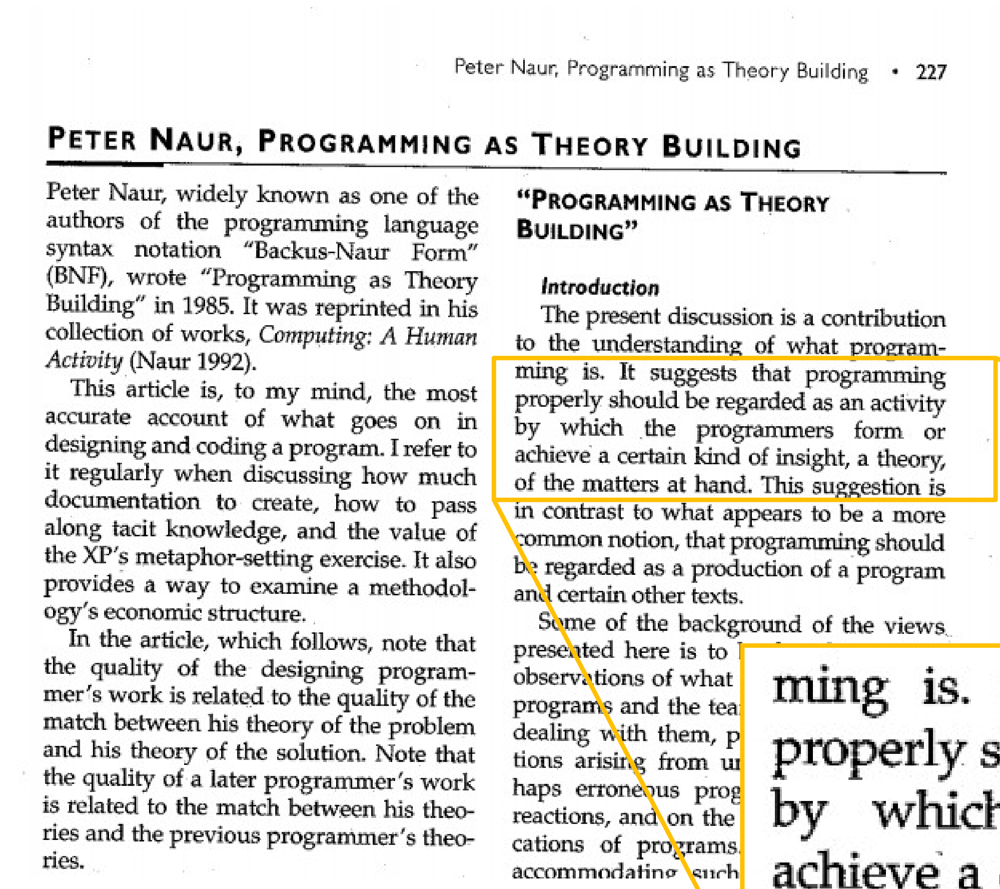
Some of the background of the views presented here is to be found in certain observations of what actually happens to programs and the teams of programmers dealing with them, particularly in situa-
tions
haps
reactic
cation
accom



match between his theory of the problem
and his theory of the solution. Note that



Programming as Theory Building



PETER NAUR, PROGRAMMING AS THEORY BUILDING

Peter Naur, widely known as one of the authors of the programming language syntax notation "Backus-Naur Form" (BNF), wrote "Programming as Theory Building" in 1985. It was reprinted in his collection of works, *Computing: A Human Activity* (Naur 1992).

This article is, to my mind, the most accurate account of what goes on in designing and coding a program. I refer to it regularly when discussing how much documentation to create, how to pass along tacit knowledge, and the value of the XP's metaphor-setting exercise. It also provides a way to examine a methodology's economic structure.

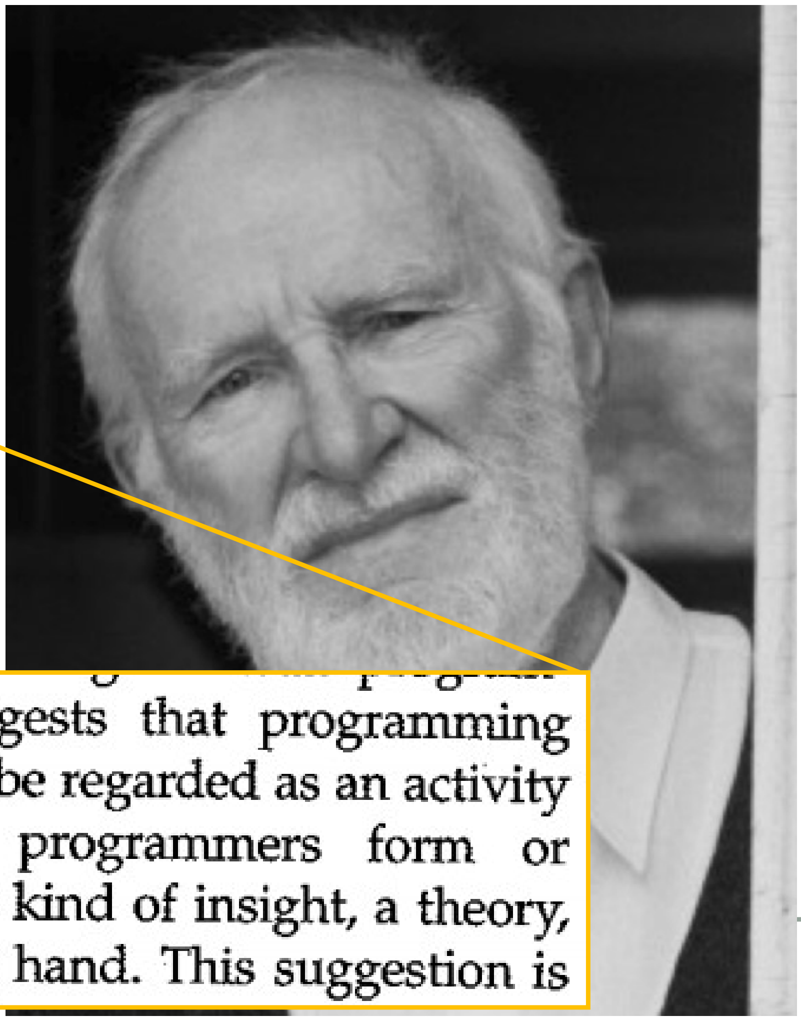
In the article, which follows, note that the quality of the designing programmer's work is related to the quality of the match between his theory of the problem and his theory of the solution. Note that the quality of a later programmer's work is related to the match between his theories and the previous programmer's theories.

"PROGRAMMING AS THEORY BUILDING"

Introduction

The present discussion is a contribution to the understanding of what programming is. It suggests that programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast to what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts.

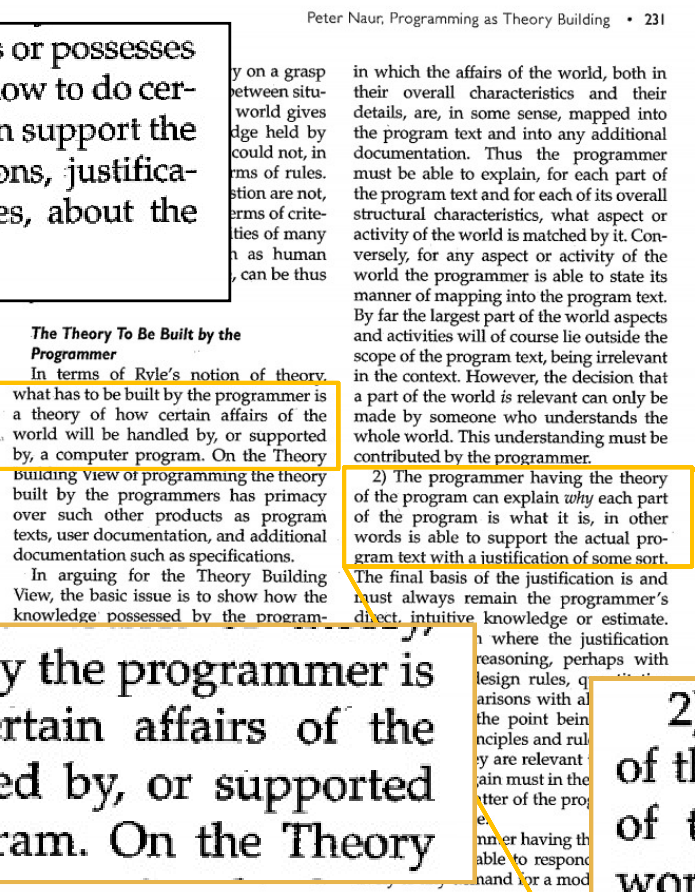
Some of the background of the views presented here is to
observations of what
programs and the tea
dealing with them, p
tions arising from u
haps erroneous prog
reactions, and on the
cations of programs
accommodating such



ming is. It suggests that programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is

Programming as Theory Building

Very briefly, a person who has or possesses a theory in this sense knows how to do certain things and in addition can support the actual doing with explanations, justifications, and answers to queries, about the activity of concern.



Peter Naur, Programming as Theory Building • 231

y on a grasp
between situ-
world gives
dge held by
could not, in
rms of rules.
stion are not,
erms of crite-
ities of many
n as human
, can be thus

The Theory To Be Built by the Programmer

In terms of Ryle's notion of theory, what has to be built by the programmer is a theory of how certain affairs of the world will be handled by, or supported by, a computer program. On the Theory Building View of programming the theory built by the programmers has primacy over such other products as program texts, user documentation, and additional documentation such as specifications.

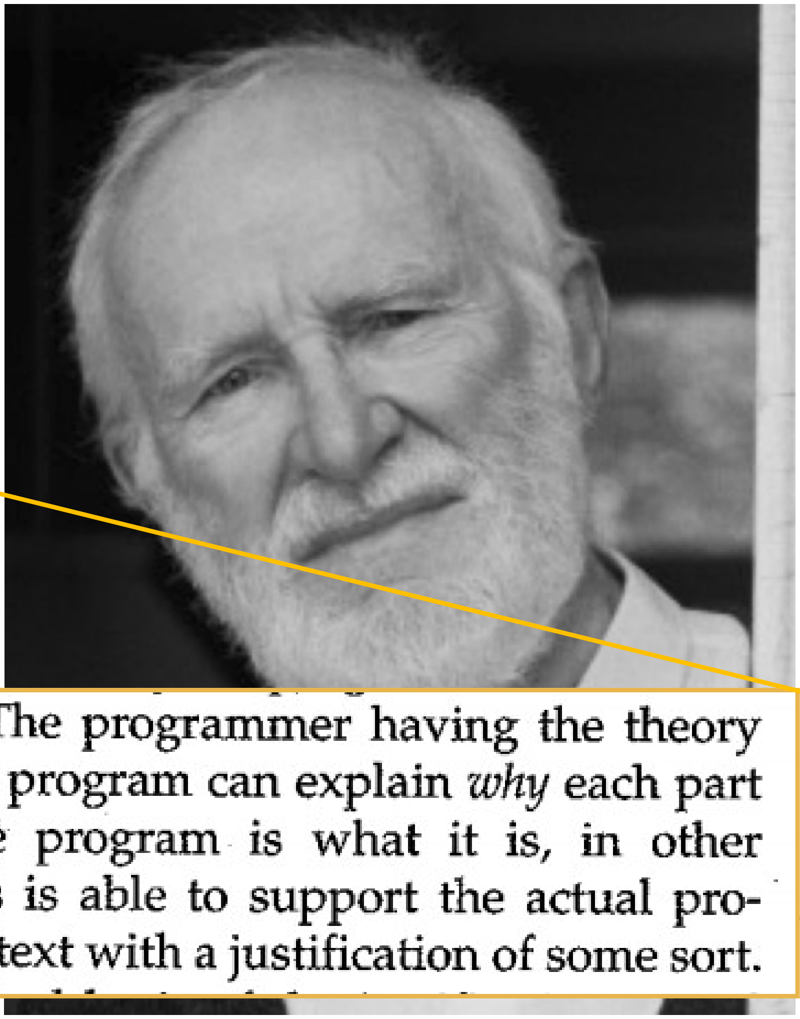
In arguing for the Theory Building View, the basic issue is to show how the knowledge possessed by the program-

in which the affairs of the world, both in their overall characteristics and their details, are, in some sense, mapped into the program text and into any additional documentation. Thus the programmer must be able to explain, for each part of the program text and for each of its overall structural characteristics, what aspect or activity of the world is matched by it. Conversely, for any aspect or activity of the world the programmer is able to state its manner of mapping into the program text. By far the largest part of the world aspects and activities will of course lie outside the scope of the program text, being irrelevant in the context. However, the decision that a part of the world *is* relevant can only be made by someone who understands the whole world. This understanding must be contributed by the programmer.

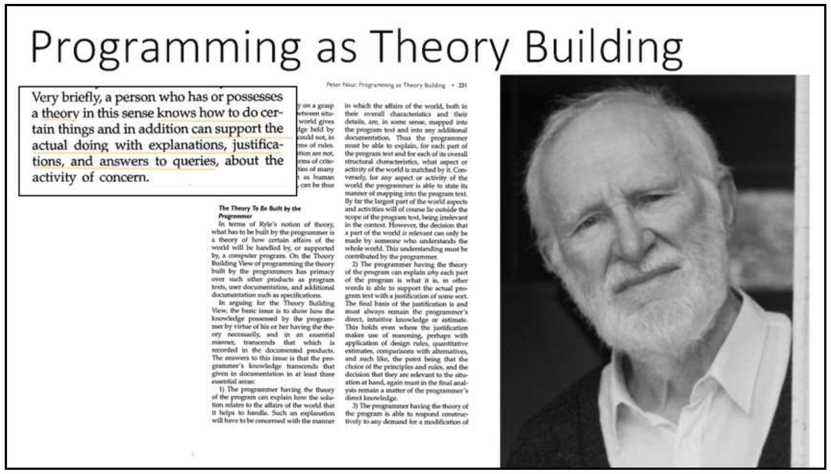
2) The programmer having the theory of the program can explain *why* each part of the program is what it is, in other words is able to support the actual program text with a justification of some sort. The final basis of the justification is and must always remain the programmer's direct, intuitive knowledge or estimate.
n where the justification
reasoning, perhaps with
design rules, q
arisons with al
the point bein
nciples and rul
y are relevant
ain must in the
tter of the pro
e.
mmer having th
able to respond
nand for a mod

what has to be built by the programmer is a theory of how certain affairs of the world will be handled by, or supported by, a computer program. On the Theory

2) The programmer having the theory of the program can explain *why* each part of the program is what it is, in other words is able to support the actual program text with a justification of some sort.



Design as Theory Building



Programming as Theory Building

Very briefly, a person who has or possesses a theory in this sense knows how to do certain things and in addition can support the actual doing with explanations, justifications, and answers to queries, about the activity of concern.

Peter Naur: Programming as Theory Building • 231

y on a grasp
etween situ-
world gives
dge held by
could not, in
rms of rules.
tion are not,
rms of crite-
ties of many
as human
can be thus

The Theory To Be Built by the Programmer

In terms of Ryle's notion of theory, what has to be built by the programmer is a theory of how certain affairs of the world will be handled by, or supported by, a computer program. On the Theory Building View of programming the theory built by the programmers has primacy over such other products as program texts, user documentation, and additional documentation such as specifications.

In arguing for the Theory Building View, the basic issue is to show how the knowledge possessed by the programmer by virtue of his or her having the theory necessarily, and in an essential manner, transcends that which is recorded in the documented products. The answers to this issue is that the programmer's knowledge transcends that given in documentation in at least three essential areas:

1) The programmer having the theory of the program can explain how the solution relates to the affairs of the world that it helps to handle. Such an explanation will have to be concerned with the manner in which the affairs of the world, both in their overall characteristics and their details, are, in some sense, mapped into the program text and into any additional documentation. Thus the programmer must be able to explain, for each part of the program text and for each of its overall structural characteristics, what aspect or activity of the world is matched by it. Conversely, for any aspect or activity of the world the programmer is able to state its manner of mapping into the program text. By far the largest part of the world aspects and activities will of course lie outside the scope of the program text, being irrelevant in the context. However, the decision that a part of the world *is* relevant can only be made by someone who understands the whole world. This understanding must be contributed by the programmer.

2) The programmer having the theory of the program can explain *why* each part of the program is what it is, in other words is able to support the actual program text with a justification of some sort. The final basis of the justification is and must always remain the programmer's direct, intuitive knowledge or estimate. This holds even where the justification makes use of reasoning, perhaps with application of design rules, quantitative estimates, comparisons with alternatives, and such like, the point being that the choice of the principles and rules, and the decision that they are relevant to the situation at hand, again must in the final analysis remain a matter of the programmer's direct knowledge.

3) The programmer having the theory of the program is able to respond constructively to any demand for a modification of

"It's developer's (mis)understanding, not [domain] expert knowledge that gets released in production"
—Alberto Brandolini

System-in-Context (use, dev, ops)	System
Developing our **theory of the problem**	Developing our **theory of the solution**
Product Design Design of system capabilities/properties	Architecture Structure and mechanisms

Design: System in Context

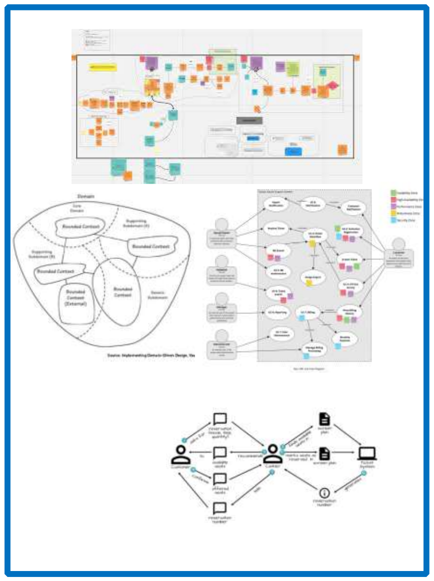
- What is the system used for (purpose and identity)?
- Which capabilities are we going to move across the system boundary?
- What new capabilities are we going to bring into existence?
- How is the system being adapted (and exapted) to new uses?

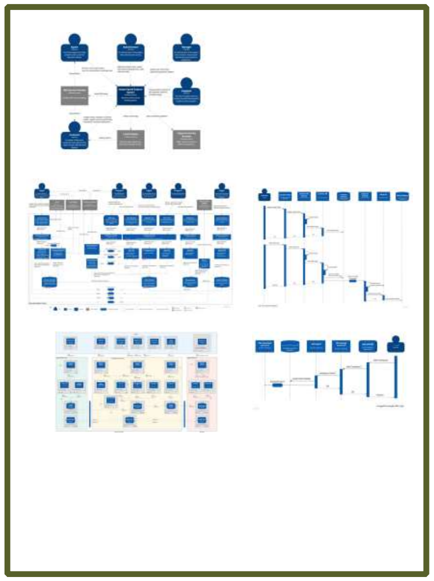
System behaviors and properties

- impact (users, partners, operations) experience

System-in-Context (use, dev, ops)

System



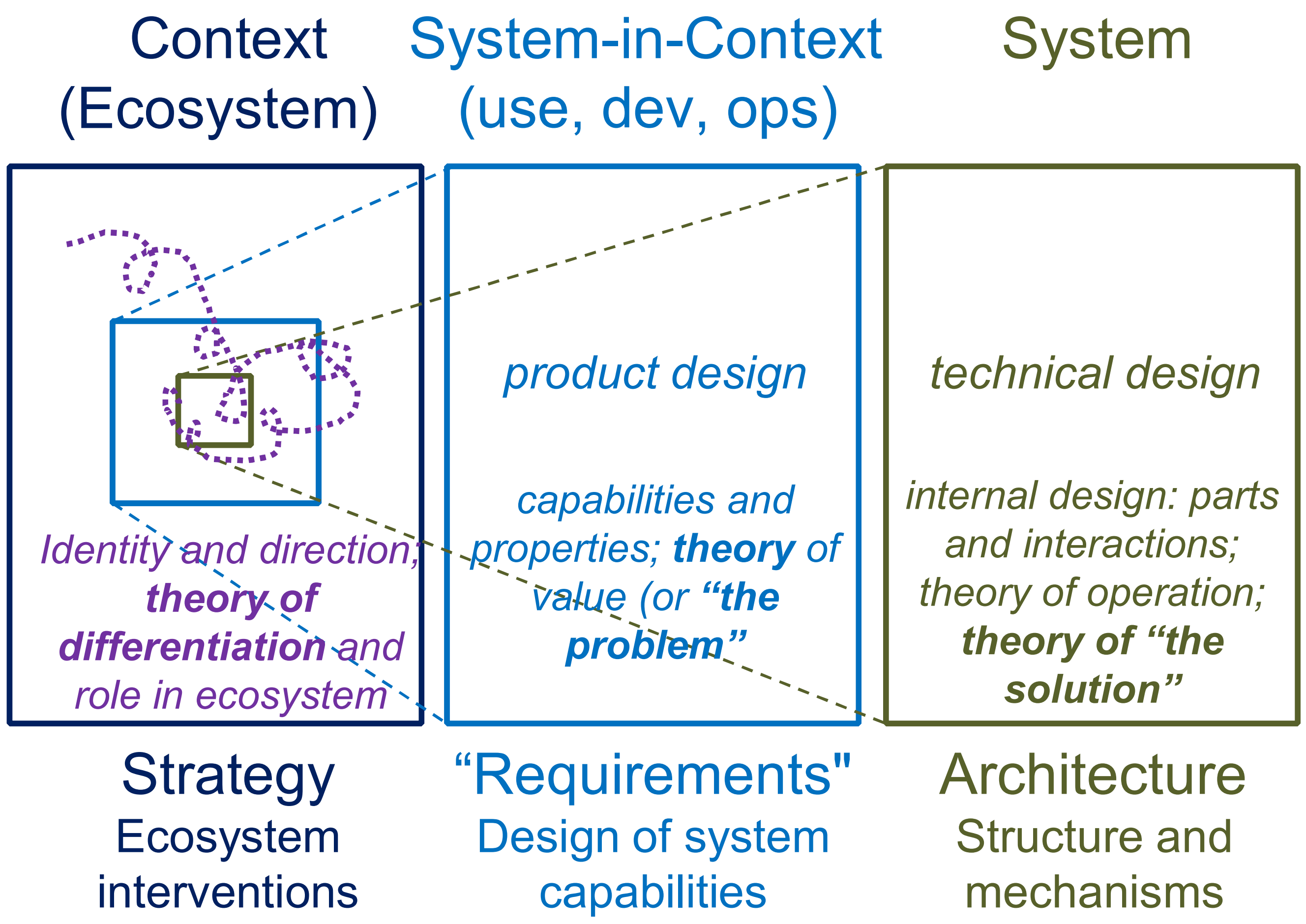


Product Design
Design of system capabilities/properties

Architecture
Structure and mechanisms

Design *Across* Boundaries

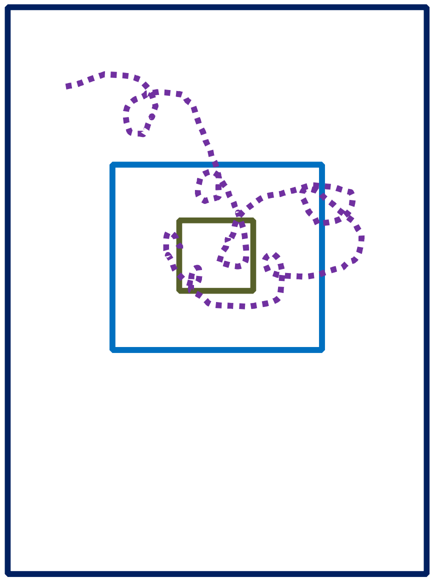
System design is contextual design — it is inherently about boundaries (what's in, what's out, what spans, what moves between), and about tradeoffs. It reshapes what is outside, just as it shapes what is inside.


Context (Ecosystem)
System-in-Context (use, dev, ops)
System
product design
technical design
Identity and direction; **theory of differentiation** and role in ecosystem
capabilities and properties; **theory** of value (or **"the problem"**)
internal design: parts and interactions; theory of operation; **theory of "the solution"**
Strategy
Ecosystem interventions
"Requirements"
Design of system capabilities
Architecture
Structure and mechanisms

Design: Nonlinear

"all models are wrong, but some are useful" – George Box

System design is contextual design — it is inherently about boundaries (what's in, what's out, what spans, what moves between), and about tradeoffs. It reshapes what is outside, just as it shapes what is inside.



Nonlinear Thinking / By Diana Montalion

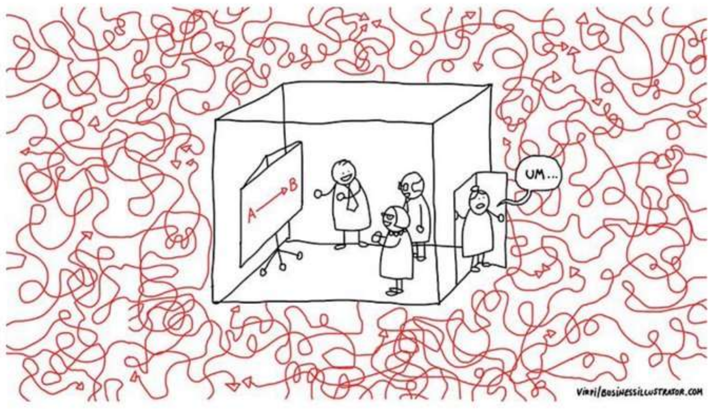
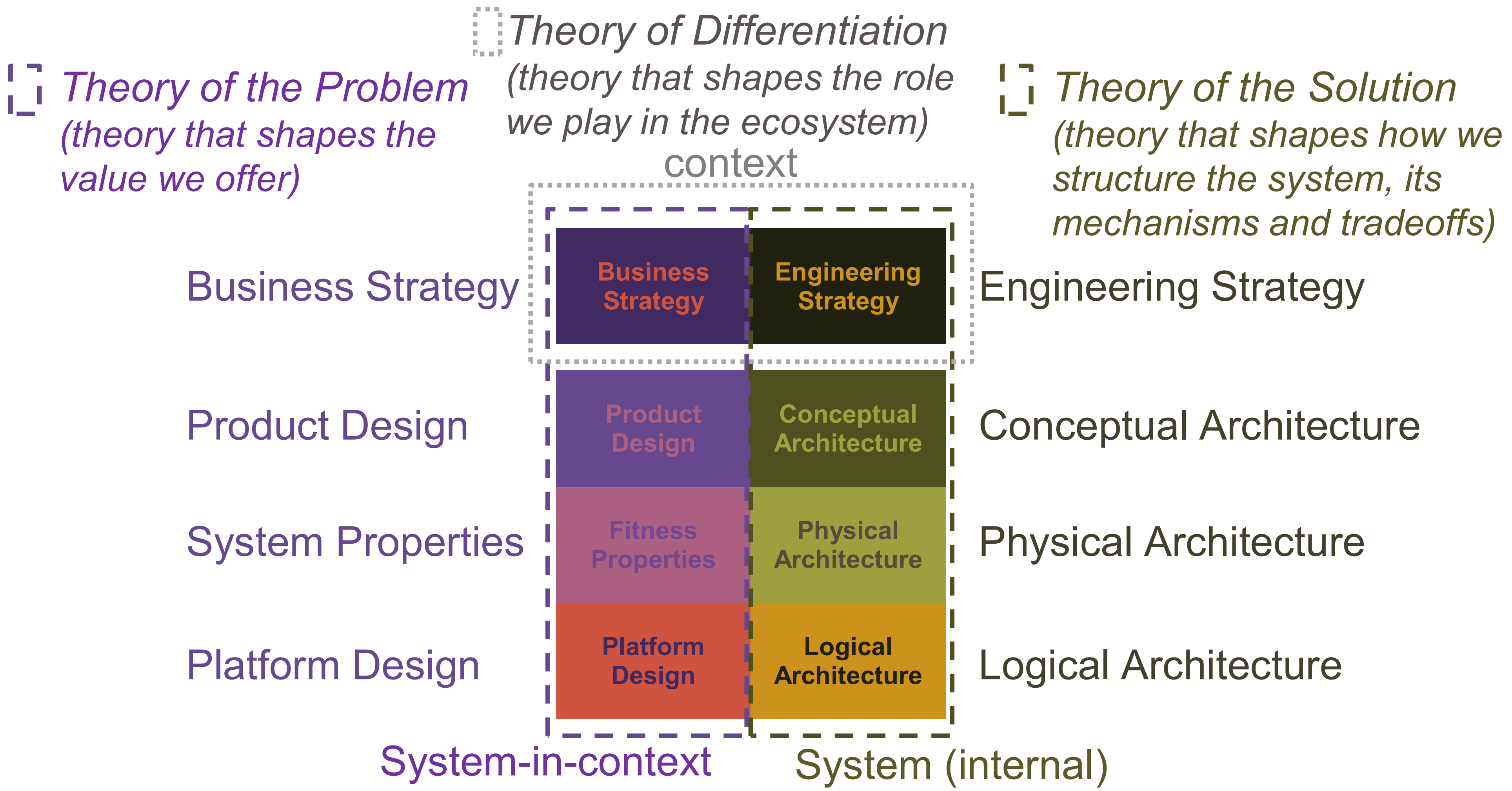


Image source: *virpi/businessillustrator.com*

Frames and Practices


Theory of Differentiation
(theory that shapes the role we play in the ecosystem)
Theory of the Problem
(theory that shapes the value we offer)
Theory of the Solution
(theory that shapes how we structure the system, its mechanisms and tradeoffs)
context
Business Strategy
Business Strategy
Engineering Strategy
Engineering Strategy
Product Design
Product Design
Conceptual Architecture
Conceptual Architecture
System Properties
Fitness Properties
Physical Architecture
Physical Architecture
Platform Design
Platform Design
Logical Architecture
Logical Architecture
System-in-context
System (internal)